计算机网络

1. HTTP--超文本传输协议

是应用层上的一种客户端/服务端模型的通信协议，由请求和响应构成，无状态

协议：规定了双方按照约定格式才能准确的通信

无状态：两次通信间没关系

**Socke**t是：TCPAPI，为了方便使用TCP/UDP抽象出来的一层，是应用层和传输控制层之间的一组**接口**，用于组织数据去符合指定协议

**WebSocket**：一个HTTP**应用层协议**--只需要一次握手然后单独建立一条TCP通信通道来传数据(用于传小数据不占很多宽带)

**CDN(内容分发网络)**：将内容缓存在终端用户附近，减少物理距离来减少时延

CDN = 更智能的镜像+缓存+流量导流

**DNS(域名系统)**:**在OSI模型的应用层，他把域名和IP互相映射成一个分布式数据库来使得人更方便地访问互联网。DNS使用TCP(区域传输的时候用)和UDP(其他时候)**

**端口53**

**过程：**向**LDNS**(本地DNS服务器)发请求用域名拿到ip，再向**权威DNS**得到答案， 权威DNS中存储着记录，最常见的3种：**A**记录（记录某域名和其IP的对应），**NS**记录（记录某域名和负责解析该域的权威DNS），**CNAME**记录（负责记录某域名及其别名）。权威能直接回答的，就回A记录；需要**其他权威DNS**回答的，就回NS记录，然后LDNS再去找其他权威DNS问；如果该记录是别名类型的，就回CNAME，LDNS就会再去解析别名

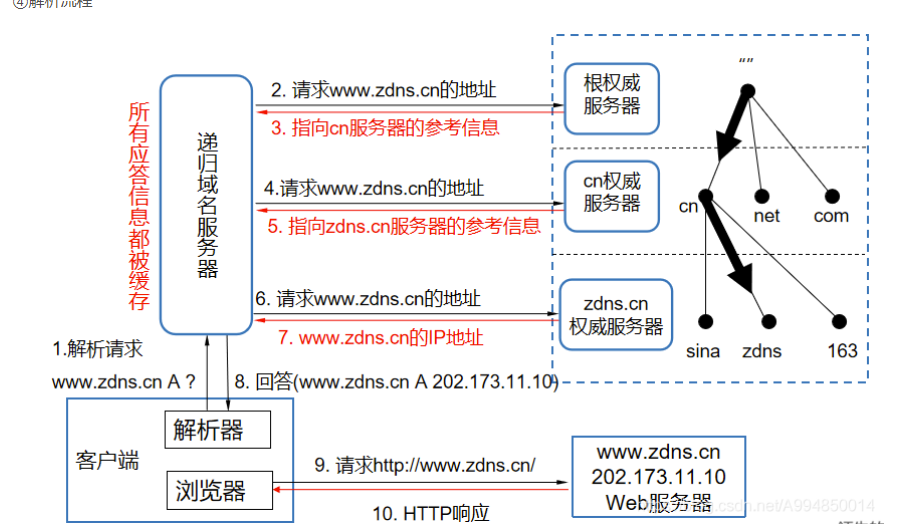
LDNS啥都不知道的时候（也即没有任何缓存），就去问**根DNS**，根能告诉LDNS下一步该问谁

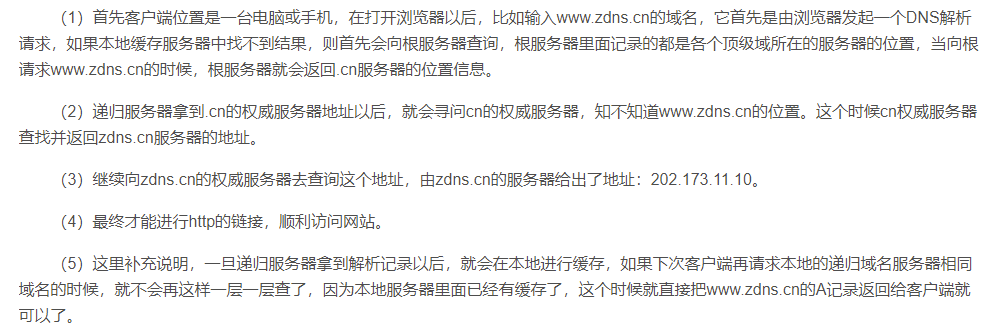
**一般DNS解析过程分为**

1、[递归查询](https://so.csdn.net/so/search?q=%E9%80%92%E5%BD%92%E6%9F%A5%E8%AF%A2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/bingo199/article/details/_blank)，即发一次请求，等你给结果，主要用于本地dns服务器

2、迭代查询，发一次请求返回一次，再重复，用于本地dns服务器向dns根域名服务器

**例子：**

****



**OSI 7层模型** 用处：规范地控制网络：

HTTP /FTP文件传输协议/webSocket Telnet SMTP TCP/UDP IP 数据链路(FDDI) IEEE 802

应用层 表示层 会话层 传输层 网络层 数据链路层 物理层

TCP/IP **5层模型及功能**：

1.物理层：将数据以实体呈现并传输(物理介质)

2.数据链路层：把数据分割成特定的可被物理层传输的帧

3.网络层：选择传送路径，通过路由找到目标主机

4.传输层：给**数据编号**、控制数据流量、查错与错误处理，确保数据可靠、**按顺序、无错地传输**　(两个主机里进程之间通信)

5.应用层：负责在网络中的**两节点之间建立和维持通信**。文件传输

为什么分层：1.各层独立，不用知道下**层怎么实现**，调整层内工作方便

2.不会有**木桶效应**，不会因某一层**技术不完善**影响整体效率

3.易于**维护**，对每层单独调试即可

**交换机**功能：1、接入层交换机支持将终端节点设备连接到网络。2、分布层交换机收集所有接入层交换机发来的数据并转发到核心层交换机。3、核心层交换机负责处理分布层传输过来的数据。

**路由器**功能：1.连通不同网络 2.选择信息传输线路

**区别**：1、交换机工作在数据链路层，根据MAC地址寻址，路由器工作在网络层，根据IP地址寻址，路由器可以处理TCP/IP协议，而交换机不可以

2、交换机是一根网线上网，但是大家上网是分别拨号，各自使用自己的宽带，大家上网互不影响。而路由器比交换机多了一个虚拟拨号功能，通过同一台路由器上网的设备是共用一个宽带账号，相互之间会受到影响。

**MAC地址和IP地址的区别**

1.MAC地址的长度为48位（6个字节），通常表示为12个16进制数，每2个16进制数之间用冒号隔开，如：00:50:29:5A:8H:1E就是一个MAC地址。

IP地址为32位，由用点分隔开的4个8八位组构成，如192.168.0.1就是一个IP地址，这种写法叫点分十进制格式。

2.所在寻址协议层上的区别

1、MAC地址应用在OSI第二层，也就是数据链路层。数据链路层协议可以使数据从一个节点传递到相同链路的另一个节点上（通过MAC地址）。

2、IP地址应用于OSI第三层也就是网络层。网络层协议使数据可以从一个网络传递到另一个网络上（ARP根据目的IP地址，找到中间节点的MAC地址，通过中间节点传送，从而最终到达目的网络）。

3.分配依据不同。

1、MAC地址的分配是基于制造商。

MAC地址由网络设备制造商生产时写在硬件内部。这个地址与网络无关，也即无论将带有这个地址的硬件（如集线器、网卡、路由器等）接入到网络的何处，它都有相同的MAC地址，是不可变的。

2、IP地址的分配是基于网络拓朴。

IP地址由网络地址和主机地址两部分组成，分配给这两部分的位数随地址类（A类、B类、C类等）的不同而不同。

**HTTP状态码**：80未加密 明文传输100—199 服务器收到要请求者继续执行请求的状态代码 200—299成功处理了请求的状态代码 300-399重定向（完成请求要其他操作） 400-499客户端出错 500—599服务端出错

常见状态码：100要客户端继续发

200服务器成功接收并处理了请求，将返回请求页面

202接受了请求，不确定处不处理

301 请求的链接发生永久性变化，客户端向新链接请求

302 请求的链接发生临时性变化 临时重定向

304 get请求被运行但请求的网页没修改

400 请求无效 401 当前用户需要验证

403 服务器得到请求但是拒绝执行

404请求失败，请求的资源没找到或者不存在

405 请求方法不对，比如要把post换成get

**URL构成**



HTTP**请求报文**：请求行(请求方法,URL,版本)，请求头(content-Type比如image/gif是gif图片格式)，空白行(通知服务器不再有请求头)，请求体(POST的数据)

HTTP**响应报文**：响应行(版本，状态码，状态码描述)，响应头

空白行，响应体(html)

常用请求头：**Accept**(可接受响应内容类型), cache-control(指定当前请求/回复**是 否用缓存**机制)

实现长连接：创建**请求方法**-设置**请求地址和参数编码格式(utf8)**-设置**长连接**-设 置请**求参数**-执行

**端口号**：

IP区分计算机，**端口号区分相同计算机的不同的服务，**一台计算机上可以同时提供很多个服务，如数据库服务、Web服务。

**客户端用IP来找服务端，用端口号来找服务端的服务，实现真正的访问**

**缓存机制**：第一次向服务器发请求，服务器返回请求的资源，之后再要请求资源就可以用**强缓存**(200，cache-control)(从本地缓存获取数据，**不**与服务器**交互**)或**协商缓存**(304，Last-Modified)(发请求到服务器，服务器判断可不可以用本地缓存)

**短连接**：客户端和服务端每次**HTTP操作就建立一次链接**，**任务结束就断开，**用于web网站的http服务因为会有很多客户端连接

**长连接：**连接后客户端或服务端关闭，客户端再次访问时继续用这个TCP连接

用于操作频繁(数据库)，点对点的通讯，而且连接数要少，因为会消耗资源

**HTTP版本**

HTTP 0.9只能发GET请求

HTTP 1.0支持**短连接**，而且会**请求部分对象**，但服务器把**整个对象**送过来了

HTTP 1.1 支持**长连接**，加了**cache-control请求头**，**宽带优化**(只送请求部分对象)

HTTP2.0:基于**https**，具有**安全性**，使用**二进制格式**和**多路复用**(1.1里只能完成上个请求才能发起下个请求，而2.0里的二进制帧可交错传输)

**HTTP传输大文件的方法**：1.数据压缩，在accept-encoding里带上gzip等格式把内容编码

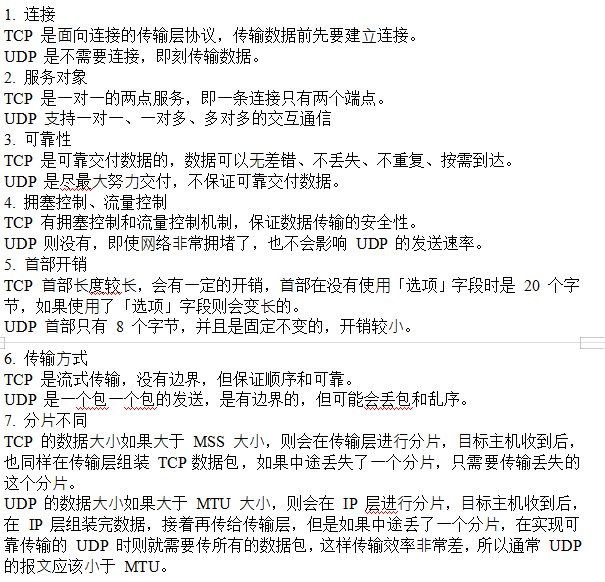
2.分块传输编码：设置Transfer-Encoding为chunked，最后一个包是空的就是传输完了。3.范围请

求,响应中存在 Accept-Ranges，可以一次性请求多个部分，206 Partial Content 状态码表示范围响应

**TCP和UDP的区别**

是传输层两个协议

**区别：**



**组成**：

UDP很简单，除了端口号几乎什么都没有，不会根据网络的情况进行拥塞控制

变可靠: 要接受方收到UDP后回复个确认包，发送方收不到确认包就要重新发

TCP除了有源端口和目标端口，还有包的序号(给包编号知道哪个先来)，确认序号(确认对方是否收到，解决丢包)，状态位(SYN发起链接，ACK回复,RST重新连接，FIN结束连接)

**TCP四元组**：源IP地址、目的IP地址、源端口、目的端口

**TCP五元组**:源IP地址、目的IP地址、协议号、源端口、目的端口

**TCP七元组**：源IP地址、目的IP地址、协议号、源端口、目的端口，服务类型以及接口 索引

**TCP唯一标识**：文件描述符，因为每一个tcp连接都要占一个[文件描述符](https://so.csdn.net/so/search?q=%E6%96%87%E4%BB%B6%E6%8F%8F%E8%BF%B0%E7%AC%A6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/tsh123321/article/details/_blank)

**应用**：

UDP：DNS的协议，看视频, 发语音, QQ聊天, 共享屏幕因为效率高但准确率低

TCP：HTTP/DNS协议，QQ传文件，邮件，登录 因为准确率高效率低

**滑动窗口作用**：用来**加速数据传输，**因为发送端希望在收到确认前，继续发送其它报文段。比如说在收到0号报文的确认前还发出了1-3号的报文，这样提高了信道的利用率。但0-4发出去后可能要重传，所以需要一个缓冲区维护这些报文，所以就有了窗口**. 同时**TCP要保证“可靠”，就需要对一个数据包进行ack确认表示接收端收到。有了滑动窗口，接收端就可以等收到许多包后只发一个ack包，确认之前已经收到过的多个数据包。有了滑动窗口，发送端在发送完一个数据包后不用等待它的ack，在滑动窗口大小内可以继续发其他数据包

**tcp缓存和窗口的关系：**窗口是缓存的一部分。

发送缓存=发送窗口+ 不允许发送的一部分

接收缓存=接收窗口+部分已确认但主机还没处理完的数据

**窗口大小指一次传输几个数据**，发送方在发送时候始终保持会一个窗口，只有窗口内的数据帧才被发送，同样接收方也有个接收窗口，只有在窗口内的才会被接收，通过改变窗口和窗口大小实现流量控制

**窗口太大**：容易**丢包**->重发->耗费带宽 **窗口太小**：**频繁**发->占**带宽**

**接收窗口大小**取决于应用(比如说tomcat：8080端口的监听进程)、系统、硬件的限制

**发送窗口的大小**swnd=min(rwnd，cwnd)。rwnd是接收窗口，cwnd用于拥塞控制，暂时可以理解为swnd= rwnd =20, 以字节为单位

当rwnd < cwnd 时，是接收方的接收能力限制发送方窗口的最大值。  
当cwnd < rwnd 时，则是网络的拥塞限制发送方窗口的最大值

**TCP保证可靠传输**：

**1.确认应答机制：**TCP传输时会把每个字节的数据都都编号，也就是序列号。在传输过程中，每次接收方收到数据后，都会对传输方确认应答。也就是发ACK报文。这个ACK报文当中带有对应的确认序列号(也叫确认号 (ack))，告诉发送方，接收到了哪些数据，下一次的数据从哪里发

**2.超时重传机制**：因为每个报文都有超时计数器，发送方在发送完数据后等待一个时间，时间超时没有接收到ACK报文，就会重发刚才发的数据。分两种情况：1.如果是数据在传输过程中由于网络原因等直接全体丢包，接收方根本没有接收到，接收方收到二次重发的数据后，就ACK应答。

2.如果是接收方接收到了响应的数据，但是发送的ACK报文响应由于网络原因丢包了，接收方发现接收的数据已经存在（判断存在的根据就是序列号，所以**序列号还有去除重复数据的作用**），那么直接丢弃，仍旧发送ACK应答。

TCP等待的时间：2MSL，等待时间过长会影响TCP传输的整体效率，

等待时间过短会导致频繁的发送重复的包

由于TCP传输时保证能够在任何环境下都有一个高性能的通信，所以这个最大超时时间（也就是等待的时间）是动态计算的

超时重传的时间 = 稍大于(往返时间+网络原因造成的抖动值)的时间

累计到一定的重传次数，TCP就认为网络或者对端出现异常，**强制关闭连接**

**3.连接管理**：三次握手四次挥手保证客户端和服务器连接的可靠性

**4.数据校验：**发送方在发送数据之前计算检验和并且填充校验和。接收方收到数据后，对数据用同样的方式计算，求出校验和，和发送方的对比。目的是为了发现TCP首部和数据在发送端到接收端之间发生的任何改动。如果接收方检测到校验和有差错，那么TCP段会被直接丢弃，重新发送。

**计算检验和**：在TCP首部加上一个12字节的伪首部。检验和总共计算3部 分：TCP伪首部，TCP首部、TCP数据，计算方式是在数据传输的过程中， 把发送的数据段都当做一个16位的整数。把这些整数加起来。并且前面的进 位不能丢弃，补在后面，最后取反，得到校验和

**5.数据合理分片和排序**：

TCP会按MTU合理分片，接收方会缓存未按序到达的数据，重新排序后再 交给应用层。

UDP在这方面的缺陷：IP数据报大于1500字节,大于MTU的时候发送方IP 层就需要把数据报分成若干片,使每片都小于MTU.而接收方IP层要重组数据 报.这样就会多做很多事情,更严重的是因为UDP的特性,当某一片数据在传送 中丢失,那么接收方就无法重组数据报.这会导致丢弃整个UDP数据报.

**6.流量控制**：是点对点通信量的控制，是端到端的问题。流量控制做的就是抑制发送端发送数据的速率，来使得接收端来得及接收。TCP会根据接收端对数据的处理能力，用滑动窗口决定发送端的发送速度的机制。如果发送端的发送速度太快，导致接收端的结束缓冲区很快的填充满了。这个时候如果发送端继续发数据，那么接下来发送的数据都会丢包，还会有一系列连锁反应，比如超时重传

**具体**：在TCP协议的报头信息当中，有一个16位字段的窗口大小，窗口字段就是用来实现滑动窗口的。窗口大小的内容实际上是接收端接收数据缓冲区的剩余大小。这个数字越大，说明接收端接收缓冲区的剩余空间越大，网络的吞吐量越大。接收端会在确认应答的时候把自己实时的窗口大小填入到报文跟ACK报文一起发送回去。发送方会根据ACK报文里的窗口大小的值的改变来改变自己的发送速度。

**如果接收到窗口大小的值为0**，那么发送方会停止发送数据。然后定期地向接收端发送窗口探测数据段，让接收端把窗口大小告诉发送端，如果接收端处理完数据了再告诉发送方开始运行

**7.拥塞控制**：拥塞控制是一个全局性的过程，涉及到所有的主机、路由器还有与降低网络传输性能有关的所有因素。就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不过载。有一个前提：网络能够承受现有的网络负荷。

当网络拥塞时，减少数据的发送,防止发送方发的太快导致过多数据注入到网络(死锁)也就是发送端发报文速度比接收端接受速度快

**拥塞：**[网络](https://so.csdn.net/so/search?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BD%91%E7%BB%9C&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41386300/article/details/_blank)中的带宽、交换结点中的缓存和处理机都是网络的资源。在某段时间，**对网络中资源的需求超过了可用的资源**，若网络中许多资源同时供应不足，网络的性能就会变坏，整个网络的吞吐量随之负荷的增大而下降。

发送方有一个拥塞窗口变量，大小=min(rwnd，cwnd)，cwnd=拥塞窗口大小, 越拥塞窗口越小

**怎么知道拥塞丢包：1.重传计时器超时 2.收到3个重复的ACK确认**

**什么时候**拥塞控制：1.网络传输TCP报文过程中发生丢失报文 2.报文需要重传

**4个拥塞机制**：

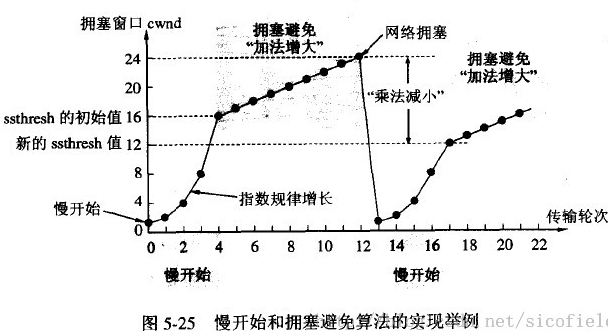
cwnd=拥塞窗口大小, ssthresh(处理拥塞时参照的一个参数,可变)

当 cwnd = ssthresh 时，慢开始算法和拥塞控制避免都可以使用

**慢开始**：当cwnd> ssthresh的时候用，思想是不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是指数增长拥塞窗口，每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1

**拥塞避免**：当cwnd< ssthresh的时候用，按线性增长拥塞窗口，比慢开始慢

重传定时器溢出的时候(也就是有很多报文没有按时发送到接收端或者接收端的ACK报文没有到达发送端)用慢开始和拥塞避免 先慢开始(指数增长)再拥塞避免(线性)



1上图是cwnd随传输轮次的变化，每过一个RTT就算一轮。

2超时就可以认为是拥塞了

**网络出现拥塞的情况**：对于慢开始和拥塞避免，只要发送方判断**网络出现拥塞**（重传器超时还没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送 方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生 拥塞的路由器有足够时间把队列中积压的分组处理完毕。

**快重传**：接收方收到一个失序报文段立即发出重复确认，而不是自己发数据时捎带确认，收到3个同样的确认就立刻重传，不等到超时

**快恢复：1.**当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。

**2.**因为发送方现在认为网络很可能没有发生拥塞，所以和慢开始不同的是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为 慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

慢开始现在只在TCP连接建立的时候和网络出现超时的时候才使用

**拥塞控制vs流量控制**：流量控制虽然可以高效可靠的传送大量的数据，但是如果在刚开始阶段就发送大量的数据，可能会导致网络拥堵，因为网络上的计算机太多了





**TCP粘包**：发送方发送的几个包数据到达接收方时粘成了一包，从接收缓冲区来看，后一包数据的头紧接着前一包数据的尾

**原因**：发送方用Nagle算法：收集多个小分组，在一个确认到来时一起发送，接收方收到数据包保存到缓存里而不是直接给应用层处理，然后应用程序再到缓存读取包。如果包到缓存的速度大于应用程序从缓存读取包的速度就会造成多个包被缓存，应用程序读到多个首尾相连在一起的包。

**解决**：发送方关闭Nagle算法，应用层循环读取数据，但每条数据要有固定格式而且发送方要把数据长度一起发送，这样就能判断开始和结束的位置。(如果发送的数据本来就是同一数据的部分就不用解决)

UDP不会粘包因为是面向消息传输的，有保护消息边界，接收方只接受一条独立地信息

TCP是基于流传输的，不认为消息是一条一条的，没有保护消息边界

**TCP拆包：**应用层每个包被拆成多个tcp报文，分别发送

解决：1.根据请求头或响应头的Content-Length决定body数据长度，用CRLF分隔。

2. 如果没有Content-Length，就用chunked协议取代Content-Length，如果请求头或者响应头有Transfer-Encoding: chunked，表示根据chunked协议读取数据。先读第一个chunk，遇到CRLF说明读取了长度，接着是第一个chunk数据，也是以CRLF结束。当遇到长度为0的chunk，表示数据读取完了

**3. 代理**

proxy代表访问用户时是代理，比如我们访问一个网址，不希望该网站知道我们的ip，就找了一个proxy，网站以为proxy的ip就是我们的ip。相反，proxy代表被访问的服务器，则此时proxy是反向代理，web-server希望对用户屏蔽一些信息就找了一个proxy隔在中间，此时proxy代表web-server集群，用户以为proxy的ip就是就是被访问web-server的ip，反向代理的服务器要有**负载均衡**功能（在多个资源中分配负载，达到最优化资源使用，避免过载）。使用服务组件Nginx实现。四层负载均衡主要工作于传输层，主要是处理消息的传递而不是内容，七层负载均衡主要工作于应用层处理消息内容。七层比四层优势：可以终止网络传输，根据消息**内容**作出负载均衡决定，CPU更密集，劣势：要为每一种应用服务专门开发一个反向代理服务器，这样限制了反向代理负载均衡的应用范围

**4. TCP三次握手**

三次握手的本质是确认通信双方收发数据的能力。

一开始，客户端和服务端都处于 close 状态。先是服务端监听某个端口，这时服务端处于 listen 状态。这个时候客户端就可以发连接请求报文了。

**第一次**客户端生成一个序列号给服务端发连接请求，报文段包含了序列号和SYN标志位=1。服务端知道客户端发件和他自己收件能力。

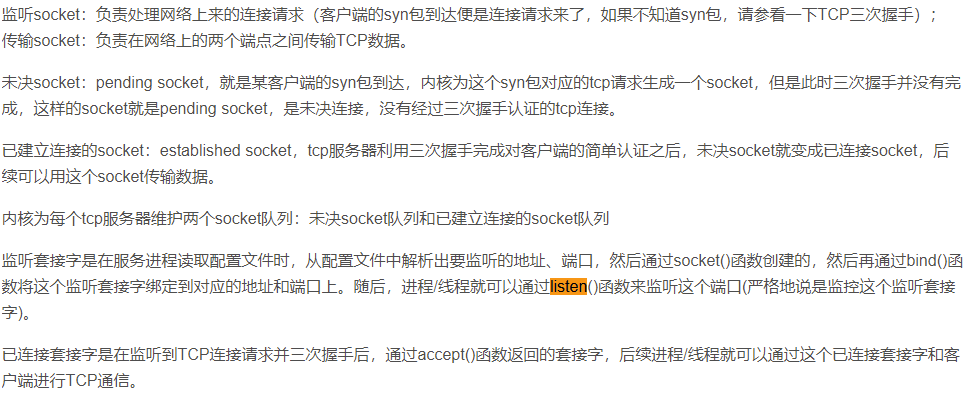
**第二次**服务端看到SYN=1是个连接请求，存下客户端序列号，之后随机成一个序列号，回复的报文里有标志位SYN=ACK=1，新序列号，以及ack=客户端序列端+1。客户端收到了，客户端知道客户端发件收件和服务端的发件收件能力OK。

**第三次**客户端收到SYN=1说明同意连接，存下服务端序列号，发送的报文里有 ACK=1，ack=服务端序列号+1，序列号=第一次握手序列号+1。服务端知道服务端发件和客户端收件能力OK

**为什么不是2次握手？**

因为要考虑连接时丢包的问题，如果只握手2次，第二次握手时如果服务端发给客户端的确认报文段丢失，此时服务端已经准备好了收发数据(可以理解服务端已经连接成功)，而客户端一直没收到服务端的确认报文，所以客户端就不知道服务端是否已经准备好了(可以理解为客户端未连接成功)，这种情况下客户端不会给服务端发数据，也会忽略服务端发过来的数据。如果有三次握手客户端发的确认ack报文丢失，服务端在一段时间内没有收到确认ack报文的话就会重新进行第二次握手，也就是服务端会重发SYN报文段，客户端收到重发的报文段后会再次给服务端发送确认ack报文。

**监听socket和传输socket**



**监听套接字是**在服务进程读取配置文件时，从配置文件中解析出要监听的地址、端口，然后通过socket()函数创建的

**监听端口**通过bind()函数将监听套接字绑定到对应的地址和端口上。之后进程/线程就可以通过listen()函数来监听这个端口(严格地说是监控这个监听套接字)

**5.四次挥手**

**第一次是**主动关闭连接的一方(客户端)，调用 close ()；协议层发送 FIN 包；

**第二次是**被动关闭的一方(服务端)收到 FIN 包后，协议层回复 ACK；然后服务端进入 CLOSE\_WAIT 状态，客户端等待对方关闭，则进入 FIN\_WAIT\_2 状态；此时，客户端等待服务端的应用程序，调用 close 操作；

**第三次是**服务端在完成所有数据发送后，调用close () 操作；这时协议层发FIN包给客户端，等待对方的ACK，服务端进入LAST\_ACK 状态；

**第四次是**客户端收到FIN包，协议层回复ACK；这时候客户端进入TIME\_WAIT 状态；服务端进入 CLOSED 状态；最后等待 2MSL 时间，客户端结束 TIME\_WAIT，进入 CLOSED 状态；

**为什么TCP连接有3次关闭时候有4次？**

因为只有客户端和服务端都没有数据发送时才能断开TCP。而客户端发FIN报文只能保证客户端没数据要发，服务端还有没有数据要发不知道。所以服务端收到客户端的FIN报文要先发一个确认报文等数据发完了再发FIN报文。所以这个不能一次性把确认报文和FIN报文发给客户端所以就多了一次

**Time\_Wait状态**：开始的时间是tcp四次挥手中主动关闭连接方发送完最后一次挥手，也就是ACK=1的信号结束后，主动关闭连接方所处的状态。Time-Wait的持续时间为2MSL. MSL是Maximum Segment Lifetime, 也就是报文最大生存时间，可为30s，1min或2min。2msl就是2倍的这个时间。工程上为2min，2msl就是4min。但一般根据实际的网络情况进行确定

**为什么客户端发出第四次挥手确认报文要2MSL（最长报文段时长的2倍）之后才能释放TCP连接？/ Time\_wait()的作用**

**作用：**为了保证客户端发送的最后一个ack报文段能够到达服务器。因为这最后一个ack确认包可能会丢失，然后服务器就会超时重传第三次挥手的fin信息报文，然后客户端再重传一次第四次挥手的ack报文。这样报文一去一回时间正好是2MSL。如果没有这2msl，客户端发送完最后一个ack数据报后直接关闭连接，那么客户端就会收到一个非法的报文段，而返回一个RST的数据报，表明拒绝这次通信，然后双方就产生异常了，那么服务器就不能按正常步骤进入close状态。也就会耗费服务器的资源。当网络中存在大量的timewait状态，那么服务器的压力机会很大。

**为什么2MSL时间**：在第四次挥手后，经过2msl的时间足以让本次连接产生的所有报文段都从网络中消失，这样下一次新的连接中就肯定不会出现旧连接的报文段了。也就是防止已经失效的连接请求报文段出现在本次连接中。

**如果没有的话就可能这样**：这次连接一挥手完马上就结束了，没有timewait。这次连接中有个迷失在网络中的syn包，然后下次连接又马上开始，下个连接发送syn包，迷失的syn包忽然又到达了对面，所以对面可能同时收到或者不同时间收到请求连接的syn包，然后就出现问题了。

**6.浏览器输入URL会输出什么**

1.浏览器**向 DNS 发送域名**，获取**IP**。先检查浏览器**缓存**有没有这个**网址映射**，有就调用，没有则去**操作系统的缓存**里找，没有则去找**LDNS**有没有，如**果LDNS仍然没有命中，就直接跳到Root Server 域名服务器请求解析**

**5. 根域名服务器返回给LDNS一个所查询域的主域名服务器（gTLD Server，国际顶尖域名服务器，如.com .cn .org等）地址**

**6. 此时LDNS再发送请求给上一步返回的gTLD**

**7. 接受请求的gTLD查找并返回这个域名对应的Name Server的地址，这个Name Server就是网站注册的域名服务器**

**8. Name Server根据映射关系表找到目标ip，返回给LDNS**

**9. LDNS缓存这个域名和对应的ip**

**10. LDNS把解析的结果返回给用户，用户根据TTL值缓存到本地系统缓存中，域名解析过程至此结束**

2.得到**IP和端口号80or443之后**，会调用系统库函数**socket**，请求一个**TCP流套接字**，**客户端向服务器发HTTP请求报文**：应用层：应用层发请求报文, 传输层：三次握手建立连续, 网络层：路由寻址,数据链路层：传输数据, 物理层：物理传输bit

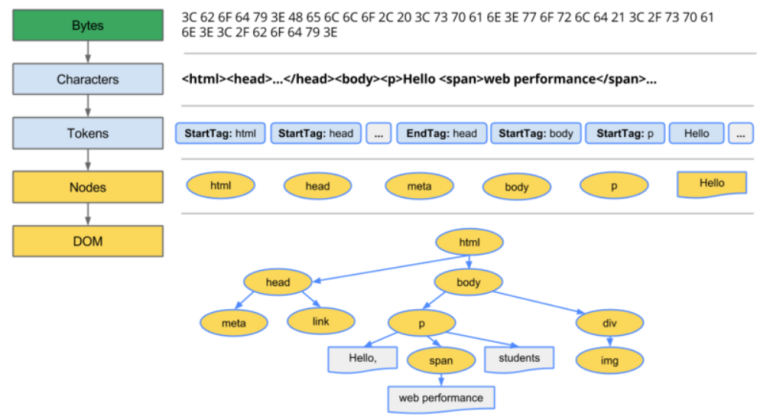
3.服务器**解析请求报文**，**发响应报文**

4.TCP**四次挥手**，关闭连接

5.客户端**解析响应报文**，浏览器开始**显示HTML**，js文件开始建立dom 树

CSS解析器解析css文件得到CSS树，浏览器结合CSS数和dom树(下载图片等)一起建立render树，这里主要是排除非视觉节点，最后渲染render树

构建DOM树：



**7.GET和POST的区别--两种发送请求方法**

HTTP最早被用来做浏览器与服务器之间交互HTML和表单的**通讯协议**；后来又被广泛的扩充到**接口格式的定义**上。底层是TCP/IP

1. **用途和幂等性**

**浏览器的GET用于获得HTML页面，图片，css，js等资源**。

**只是读取数据 -> 可以对数据做缓存（做到浏览器 代理 服务端都可以）->幂等(副作用相同) 幂等就是多次操作副作用相同**

**POST用于更新资源信息 ->不是幂等(副作用不同)->不能多次执行和不能缓存**。

比如一个下单页面 ->下单成功被存到缓存，下次下单就不会对服务器发送请求 ->直接返回缓存的下单成功页,导致没有下单。当然get也可以有副作用 post也可以没有。

**2.数据位置**

GET的数据附在**URL之后的queryString传输**，更加容易被看到，更不安全

POST：请求的**body**传输数据，安全一些（但是HTTP实现**接口**发送请求时（ajax，postman发出请求）参数可以放任何位置比如header body querystring..这种不确定性会造成低效所以用了REST规定了增删改查)

RESTFUL规范：

1.url链接设计：采用https方式，有api关键字，有版本需要明确版本，请求链接用名词来表示资源，具体的操作方式采用请求方式来确定  
2.url响应数据设计：需要明确 状态码、错误信息、成功结果，子资源一般用子资源的接口来标注

但是私密数据在body里也是会被记录的，所以都不安全，所以要通过https

**3.数据大小**

GET方法传输的数据量是**2KB**而且在URL种传送的参数有**长度限制**，POST都没有，但真正影响还是**服务器和HTTP的规定**

**4.请求缓存时**

GET请求类似于**查找**过程，用户获取数据，**不用每次和数据库连接**，POST一般是修改和删除的工作, **必须和数据库交互 ->** GET适合请求缓存

5.GET产生一个TCP数据包，POST产生2个:

GET请求，浏览器会把http header和data一起发送，服务器响应200，

POST请求，先发http header,服务器响应100continue，再发data，服务器响应200

但是GET和POST本质就是**TCP链接**，并无差别，是HTTP的规定和浏览器的限制导致他们不同

**8.http请求方法**

GET请求**页面信息**

POST向指定资源**提交数据进行处理**

PUT传送的**数据取代**指定**内容**

DELETE**删除**指定**页面**

OPTIONS允许客户端查**看服务器性能**

HEAD类似GET，但返回的**响应没有内容只有报头**

TRACE**显示**服务器收到的**请求**，用于测试或诊断

**9.http和https的区别？**

http端口号80，运行于TCP之上，**明文传输**，客户端和服务端**无法验证**对方**身 份，资源消耗少**，**安全性低**

https端口443，HTTP协议+SSL协议，是添加了加密和认证 的http，加密通过对称加密和非对称加密混合加密 -> 消耗更多CPU和内存，需要花钱买**证书**(分DV, OV, EV三种类型 可信度越来越高，越来越严格)

**为什么混合加密**：

数据是被对称加密传输的，对称加密过程需要客户端的一个密钥，为了确保能把该密钥安全传输到服务器端，采用非对称加密对该密钥进行加密传输，总的来说，对数据进行对称加密，对称加密所要使用的密钥通过非对称加密传输。

**过程：1.客户端向服务器发起HTTPS请求，连接到服务器的443端口**

**2.服务器端有一个密钥对，即公钥和私钥，是用来进行非对称加密使用的，服务器端保存着私钥，不能将其泄露，公钥可以发送给任何人。**

**3.服务器将自己的公钥发送给客户端。**

**4.客户端收到服务器端的证书之后，会对证书进行检查(看签发机构可不可信任)，如果发现证书有问题，那么HTTPS传输就无法继续。如果公钥合格，那么客户端会生成一个用于进行对称加密的密钥。然后用服务器的公钥对客户端密钥进行非对称加密，这样客户端密钥就变成密文了，至此，HTTPS中的第一次HTTP请求结束。**

**5.客户端会发起HTTPS中的第二个HTTP请求，将加密之后的客户端密钥发送给服务器。**

**6.服务器接收到客户端发来的密文之后，会用自己的私钥对其进行非对称解密，解密之后的明文就是客户端密钥，然后用客户端密钥对数据进行对称加密，这样数据就变成了密文。**

**7.然后服务器将加密后的密文发送给客户端。**

**8.客户端收到服务器发送来的密文，用客户端密钥对其进行对称解密，得到服务器发送的数据。第二个HTTP请求结束**

**对称加密**加密解密用的同一个密钥，传输过程会被截获，不安全，但传输快

其加密过程如下：**明文 + 加密算法 + 私钥 => 密文**  
解密过程如下： **密文 + 解密算法 + 私钥 => 明文**

由于对称加密的算法是公开的，所以一旦私钥被泄露，那么密文就很容易被破解，所以对称加密的缺点是密钥安全管理困难。

**非对称加密**用公钥加密，私钥解密，公钥无法反推导出私钥所以安全，但加密解密废时间，速度慢

被公钥加密过的密文只能被私钥解密，过程如下：  
明文 + 加密算法 + 公钥 => 密文， 密文 + 解密算法 + 私钥 => 明文  
被私钥加密过的密文只能被公钥解密，过程如下：  
明文 + 加密算法 + 私钥 => 密文， 密文 + 解密算法 + 公钥 => 明文

**SSL**是安全套接层协议**TLS**安全传输层协议

**10.Session和Cookie**

**Cookie**是服务器保存在浏览器上的key-value格式文件，有用户信息

最大作用：存储sessionId来唯一标识用户

**Session依赖**Cookie实现, 是**服务器分配的一块储存空间**

**过程**：客户端向服务器发请求，如果**服务器需要记录该用户状态**，就颁发浏览器一个Cookie，并且自己创建一个session -> **客户端**浏览会把Cookie**存**起来，再请求该网站时，浏览器会把请求网址和Cookie一起发给服务器。服务器检查该Cookie来里的sessionID来辨认用户身份。

**cookie、session(放敏感信息)与localStorage区别**

1. 位置

cookie数据在客户端，安全性低 session数据在服务器，安全性高

1. 大小

Session和localStorage比cookie大得多

1. 存在时间

localStorage存储持久数据，只能手动删

sessionstorage关浏览器

cookie设置了过期时间，浏览器关了也不会删，没设置则关闭浏览器后失效

session有失效时间，如果session倍访问，失效时间归0开始计

session会比较占用服务器性能，当访问增多时应用cookie

**token**

cookie和token都是**首次登录**时由服务器**下发用来验证**

**好处：不用账号密码 -> 减少数据库查询**

**区别(管理、共享)**：token由**应用管理**, 可以**避开csrf**攻击并能在多个**服务间共享**

**csrf**：浏览器访问服务器后得到cookie，之后访问非法网站之后，cookie和里面的ticket(凭证)被盗取。之后非法网站模拟你的身份用post表单向服务器提交数据。

如果提交的是转账的数据就很危险有可能将钱划走

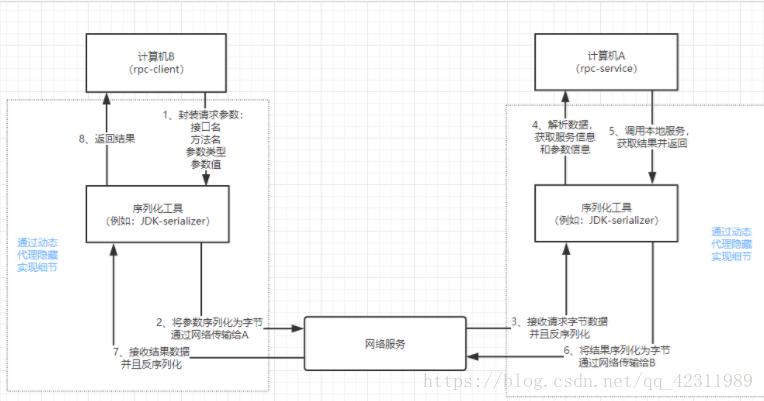
**Spring security解决方案**：会在返回给浏览器的表单里添加一个tocken，tocken每次请求都不一样，是随机生成的，非法网站能窃取cookie但不能猜到tocken，所以post提交数据的时候，服务器看到ticket对但没有tocken则判断为csrf攻击，提交失败（但security只能在非异步请求，在返回网页时自动传一个tocken，异步时需要咱们自己处理，前端模板上编写逻辑加入tocken ）

Unicode 16进制 UTF-8二进制 UTF-8是Unicode的一种实现

**RST标志位**表示复位，用来异常的关闭连接。用RST包关闭连接时不用等缓冲区的包都发出去（不像FIN），直接丢弃缓冲区的包发送RST包，接收端收到RST包后也不用发ACK包确认。比如A向B连接，但B没有监听相应地端口，B的TCP处理程序就会发RST包

**RST攻击**：AB建立了连接，Ｃ伪造一个TCP包发给B，让B丢弃缓冲区的数据而且异常地和A断开连接。想伪造A的包要知道A的源IP、源端口、目标IP、目标端口

**RPC**是远程过程调用，也就是在一个服务器的应用要调用另一个服务器上的应用的方法，但不在一个内存空间，不能直接调用，只能靠网络来调用



**Linux指令**

scp，管道传输：Linux主机之间传输文件

ps -A：显示**所有**程序

ps -ef：用**标准格式**显示进程

ps aux:用BSD的**格式**显示进程

file：**判断**文件类型

fd：查看已被打开的文件描述符

grep walkpgdir 文件名：print出文件中**匹配行内容**

grep -i :不区分大小写print出文件中匹配行

find：在目录重搜索文件，有则print出文件名

top：CPU使用状态，内存使用情况

vi替换字符串：s/vivian/sky 替换**当前**行**第一个**vivian为sky

s/vivian/sky/**g** 替换**当前**行**所有**vivian为sky

%s/vivian/sky/g 替换**所有行**所有vivian为sky

more **分页展示**文件内容

less**分屏展示**内容，根据**显示需要加载内容**，对**大型文件**效率更高

head -n 5查看文件头5行， tail反之。

**ifconfig:显示网络接口，子网掩码**

netstat：看网络情况

netstat -anpt | grep 80：查看80端口 – 与网络有关用netstat 再用grep找端口

Lsof -i:端口号 ：查看端口被什么进程占用

ps aux | grep sha(程序的名字)：查线程号

ps -ef |grep mysql 显示有关mysql的进程

查看日志：tail/head来显示test.log内容 + cat -n test.log | grep “debug”查询关键字 的日志

vim进入log文件用/+关键字查找，下一个按n就行

kill -9 pid：杀死进程 + 重新加载和停止进程，发信息 9是SIGKILL(发给进程)

**SSH**是一种网络协议，用于计算机之间的加密登录

用于远程登录上Linux主机。

常用格式：ssh [-l login\_name] [-p port] [user@]hostname  
更详细的可以用ssh -h查看。

不指定用户，默认使用root账户登录：ssh 192.168.0.11

指定用户：ssh -l root 192.168.0.11 | ssh root@192.168.0.11

**Git指令**

git branch查看本地所有分支

git rm 文件名 删除指定文件

git log查看commit日志

git clone克隆版本库

把本地文件/文件夹 移到github上：

git config --global user.name “” --设置github用户信息

git config --global user.email “”

git init 初始化 --> 生成一个 .git文件夹

git add . 添加当前文件夹下所有文件

git commit -m “”提交到Repository

git remote add origin 加上仓库地址

git push origin master将文件推到服务器上

merge和rebase用于合并分支

区别：1.merge**不会**保留merge分支的**commit到日志**

2.**处理冲突(不知道要保留什么设置)**时，merge**会产生一个commit**，rebase不会

版本回退：

1.没用git add.缓存代码：git checkout — filepathname

2.用了git add：用git reset HEAD filepathname

3.已经提交了代码：git log看git提交历史，里面第一行就是commitid，然后用这个id去git reset —hard commitid回退

4.已经push到远程服务器：git reset —hard

git push origin HEAD —force

**版本控制**：是记录软件开发过程里文件内容的变化用来查特定版本修订情况的系统

**好处**：

1. 记录什么人什么时候改了什么内容，每一次文件的改变，文件的版本号都将增加，确保在开发过程中由不同人所编辑的内容都得到更新

2. 方便并行开发，因为可以有效地解决版本的同步以及不同开发者之间的开发通信问题，并行开发中最常见的不同版本软件的(Bug修正问题也可以通过版本控制中分支和合并的方法解决

**版本控制系统**：

Git(分布式) 客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。这么的话，任何一个服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复

SVN/ CVS(集中式) 一个单一的集中管理的中央服务器，它保存了所有文件的修订版本，然后一起工作的人都可以通过客户端连到这台服务器，拿到最新的文件或者提交更新

Python(一切都可看做对象)

1. **多进程和多线程的区别**

1、多线程可以共享全局变量，多进程不能。

　　2、多线程中，所有子线程的进程号相同；多进程中，不同的子进程进程号不同。

　　3、线程共享内存空间；进程的内存是独立的。

　　4、同一个进程的线程之间可以直接交流；两个进程想通信，必须通过一个中间代理来 实现。

　　5、创建新线程很简单；创建新进程需要对其父进程进行一次克隆。

　　6、一个线程可以控制和操作同一进程里的其他线程；但是进程只能操作子进程。

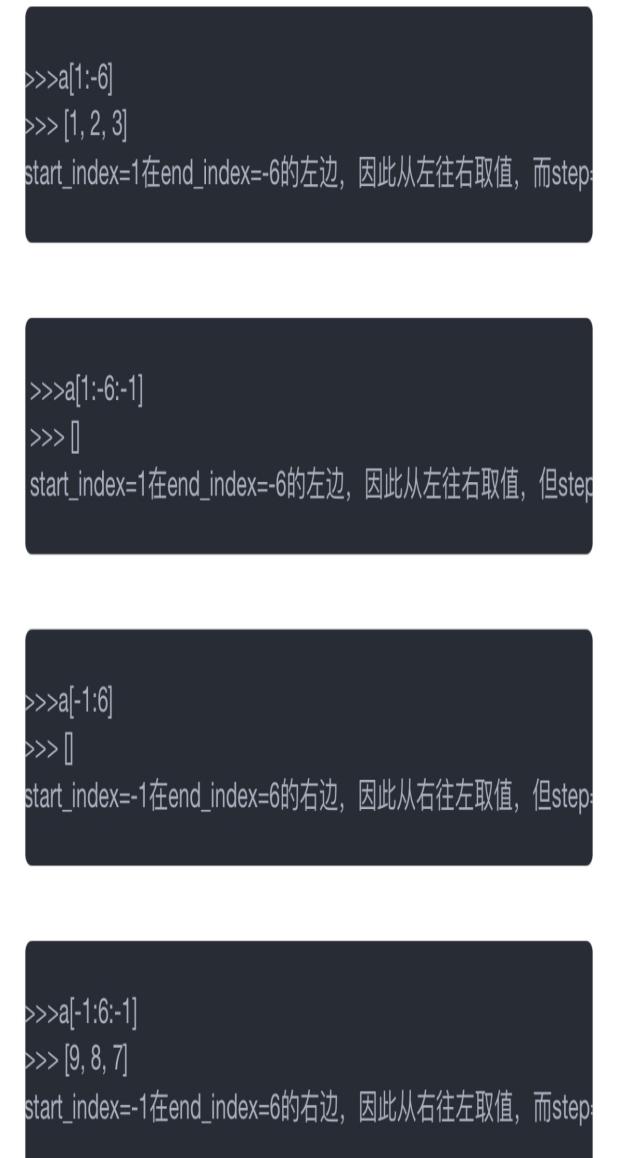
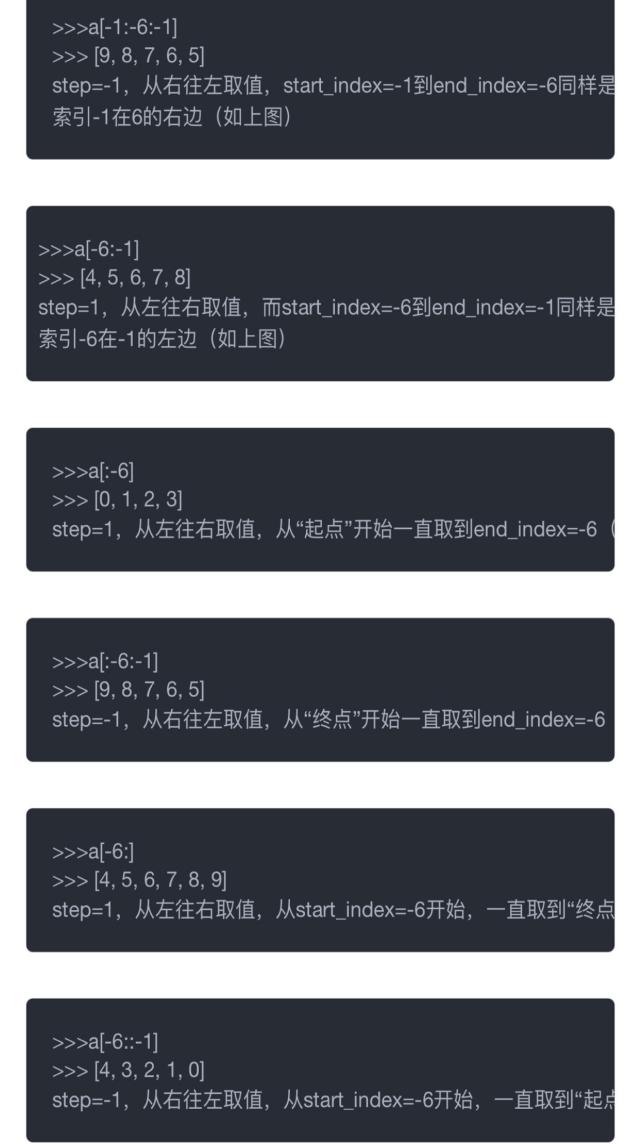
　　7、两者最大的不同在于：在多进程中，同一个变量，各自有一份拷贝存在于每个进程 中，互不影响；而多线程中，所有变量都由所有线程共享。

1. 去首尾空格：strip，去掉左空格：lstrip，去掉右空格：rstrip
2. list：有序集合，靠索引增删insert，pop(index),append和pop()是对list末尾增减。
3. tuple：有序列表，只能用[]获取元素，只能用[][]改tuple里的list的元素
4. 切片startindex：endindex：step，

start省略时，step为正则从起点开始，为负 从终点开始

end省略时，step为正时取到终点，为负时取到起点

取偶数位[::2]取奇数位[1::2]



1. 浅拷贝：只拷贝数据集合最外边的一层，深层的数据只是做了内存地址引用，并没有拷贝。假如a是一个包含数组的数组，b浅拷贝了a，则a改了他的数组元素里的数组元素，b也会相应地改。

深拷贝：完全拷贝数据集合的所有数据，与源数据再无相关。



1. 常用库：numpy(科学计算包) beautifulsoap(爬虫相关)

requests (HTTP请求库) pip(安装和管理python包的工具)

1. 在运行时才确定对象的类型和内存--动态类型 (不像Java，C++)
2. 内存管理：用一个私有Heap来放所有对象和数据结构，我们无法访问

引用计数：x=3.14,x是第一个引用,y=x，y是第二个引用，计数=2

垃圾回收：清楚引用计数=0的对象的内存和互相引用的两个对象

用id得到对象身份标识，is比较对象地址

1. 数据类型：不可变：Number，String，tuple

可变：List，dictionary，set

a=1 b=a b+=1 a还是1

不可变是指b=a会分配b一个新的内存，并把a的值复制给b

可变指=后两个对象指向同一个内存

10. 内存管理：创建大量消耗小内存的对象，频繁调用malloc会导致很多内存碎片，减低效率，内存池就是先在内存申请一定数量、大小相等的内存块备用，有内存需求先从内存池分配，不够再申请，这样就能减少内存碎片，提升效率

小于256bits ，pymalloc申请内存

大于256bits，new/alloc申请内存

回收：对象引用计数=0，从内存池来的到内存池去

**读文件**的过程1、进程调用库函数向内核发起读文件请求；

2、内核通过检查进程的文件描述符定位到虚拟文件系统的已打开文件列表表项；

3、调用该文件可用的系统调用函数read()

3、read()函数通过文件表项链接到目录项模块，根据传入的文件路径，在目录项模块中检索，找到该文件的inode；

4、在inode中，通过文件内容偏移量计算出要读取的页；

5、通过inode找到文件对应的address\_space；

6、在address\_space中访问该文件的页缓存树，查找对应的页缓存结点：

1.如果页缓存命中，那么直接返回文件内容；

2.如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过inode找到文件该页的磁盘地址，读取相应的页填充该缓存页；重新进行第6步查找页缓存

**写文件**

前5步和读文件一致，在address\_space中查询对应页的页缓存是否存在：

6、如果页缓存命中，直接把文件内容修改更新在页缓存的页中。写文件就结束了。这时候文件修改位于页缓存，并没有写回到磁盘文件中去。

7、如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过inode找到文件该页的磁盘地址，读取相应的页填充该缓存页。此时缓存页命中，进行第6步。

8、一个页缓存中的页如果被修改，那么会被标记成脏页。脏页需要写回到磁盘中的文件块。有两种方式可以把脏页写回磁盘：

1. 手动调用sync()或者fsync()系统调用把脏页写回

2. pdflush进程会定时把脏页写回到磁盘

同时注意，脏页不能被置换出内存，如果脏页正在被写回，那么会被设置写回标记，这时候该页就被上锁，其他写请求被阻塞直到锁释放